Optimal Partitions of Data in Higher Dimensions

Bradley W. Jackson

Department of Mathematics, San Jose State University

jackson@math.sjsu.edu

Jeffrey D. Scargle

Space Science Division, NASA Ames Research Center

Jeffrey.D.Scargle@nasa.gov

and

Chris Cusanza, David Barnes, Dennis Kanygin,
Russell Sarmiento, Sowmya Subramaniam, Tzu-Wang Chuang

San Jose State University, Center for Applied Mathematics and Computer Science

Nov. 19, 2003

ABSTRACT

Given any collection of data cells in a data space X, consider the problem of finding the optimal partition of the data cells into blocks which are unions of cells. The algorithms we describe can be used to find the optimal partition of a set of data cells in any dimension. These algorithms work for any objective function that is strictly convex and additive on the blocks of a partition. We describe an $O(N^2)$ dynamic programming algorithm for finding the optimal partition of N cells into arbitrary blocks (not necessarily connected) and we also give a branch and bound algorithm for finding the optimal partition of N cells into connected blocks. These results can be used to search for clusters in data, signal processing, classification of data, image processing, density estimation, pattern recognition and in a variety of other applications.

Subject headings: signal processing, galaxy clusters, data analysis, algorithms, dynamic programming, branch and bound

1. Introduction

In time-series analysis, one often has a set of data points on a time interval to represent the varying intensity of the signal from a source with unknown properties. We also have the following problem in astronomical data analysis. Data points in dimension 2 or 3 can be used to represent the positions of galaxies in space. In either case, we want to partition the data points (galaxies, etc.) into regions that are roughly uniform in density. The high-density regions might represent galaxy clusters or other interesting phenomenon. We start with a collection of cells containing the data points and consider partitions of the cells into blocks that are unions of cells. Our goal is to find the optimal partition of the data.

In general, suppose we are given a set of data points in a bounded region X of $\Re_n$ and let C be a set of N cells whose union is equal to X. We often use the Voronoi diagram of a set of data points (see Figure 1), which contains one cell for each data point, as our set of data cells. In a Voronoi diagram each point in the data space is assigned to the cell containing the data point that it is closest to, thus the data cells are determined by the data. A block B is defined to be any union of cells from the set C.

For a given set of data points our goal is to find the best piece-wise constant function that represents the data. Each partition of the data into blocks has a corresponding piece-wise constant function, that is constant on the blocks of the partition. To quantify what we mean by the best partition we assign a numerical value to each partition and then try to solve the resulting combinatorial optimization problem, by finding the partition which has the maximum value. Such a quantity goes by many names depending on the application: goodness of fit, objective function, fitness, and many others. Here we simply use the generic term "value", and for example refer to the value of a partition or of a block (since we see below that the value of a partition is sometimes defined using the values of its blocks). This quantity measures how well the corresponding piece-wise constant model (constant on the blocks making up the partition) fits the data. This can be implemented by maximizing some measure of model fitness, such as the posterior probability of the model, given the data. As described elsewhere (Scargle 1998), by marginalizing the model parameters we get a value that depends only on the blocks and not on the locations of the data points. For any block, B, in P, we denote its area (or volume or length) by $a(B)$, the population of block B by $n(B)$, the number of data points in block B, and the "average" density of block B by $n(B)/a(B)$. Under suitable assumptions (Scargle 1998) we were able to assign the posterior probability (likelihood) of a Poisson distribution with constant density (equal to the average density) over a region of area $a(B)$ containing $n(B)$

data points, equal to

$$f(a(B), n(B)) = \beta(a(B) - n(B) + 1, n(B) + 1) \quad (1)$$
$$= \Gamma(n(B) + 1) * \Gamma(a(B) - n(B) + 1)/\Gamma(a(B) + 2). \quad (2)$$

This formula holds for data in any dimension (the definition of $a(B)$ changes to the appropriate measure of volume for the given dimension). The likelihood of a given partition is the product of the likelihoods over all the blocks in that partition since we assume that the probabilities on each region are independent of each other. Thus the best (most likely) subpartition is one which maximizes

$$V = \prod f(a(B_i), n(B_i)). \quad (3)$$

We refer to a partition which achieves the maximum value as an optimal partition. Note that a partition which maximizes V also maximizes W = log V,

$$W = \sum g(a(B_i), n(B_i)), \quad (4)$$

where $g(a(B_i), n(B_i)) = log f(a(B_i), n(B_i))$. Thus our goal is to find a partition P_{max} which maximizes $W = \sum g(a(B_i), n(B_i))$ where the sum is taken over all the blocks in the partition.

2. Finding Optimal Partitions in Dimension 1

Suppose that g is a function that assigns a value to any block and for any partition P, the value of P, $W(P)$ is equal to the sum of the values of its blocks, $\sum g(a(B_i), n(B_i))$. In this case we say that W satisfies the additive property. Let P_{max} be any optimal partition with respect to W, and let P_0 be any subpartition of P_{max}. It follows from the additive property that P_0 is an optimal partition of the set that it covers. This is known as the principle of optimality (Bellman 1957).

Using the principle of optimality we were able to show that dynamic programming (Jackson,Scargle,et.al. 2003) gives an $O(N^2)$ algorithm for finding the optimal partition of N data points on an interval. In addition, to being useful in time-series analysis this dynamic programming algorithm can also be applied to find optimal histograms for data (with unequal bins allowed), for a variety of signal processing tasks, etc.

In practical terms, the principle of optimality states that once the optimal partitions of the first j cells, $j = 0, 1, 2, \ldots, i$ are found, the optimal partition of the first

$i+1$ cells can be found by determining which of the the following $i+1$ partitions has the maximum value. For $j = 0, 1, 2, \ldots, i$ consider the optimal partition of the first j blocks together with a single block containing cells $j+1, \ldots, i+1$. Using the principle of optimality we see that the partition with the maximum value in this group will be the optimal partition of the first $i+1$ cells. To compute the optimal partition of the first $i+1$ cells we need to find $i+1$ values and compute their maximum which takes $O(i+1)$ steps and requires $O(i+1)$ values to be stored. Thus the total number of steps required is $O(1+2+\ldots+N) = O(N^2)$ and the maximum amount of memory required is $O(N)$. The incremental way that this algorithm operates on the data also allows it to operate nicely in an on-line mode (performing calculations on the first i data points as we are waiting for the next data point to be transmitted). This mode has been found to be very useful in the rapid detection of changepoints in a data stream.

Dynamic programming has also been shown to be an efficient technique for finding the optimal solution for a variety of other 1-dimensional data analysis problems (Hubert 1997; Kay 1998; Kehagias,Nicolau,Fragkou,Petridis 2004; Quintana,Iglesias 2003; Vidal 1993). In most cases one seeks the optimal partition into k blocks, for some fixed k. However, our algorithm is able to compare partitions with different numbers of blocks, so the number of blocks is automatically determined by the data.

Relatively little has appeared about finding the optimal partition of a set of data points in higher dimensions. Indeed, for many standard problems in higher dimensions it is known that the problem of finding the optimal partition is NP-complete. Unlike the situation in dimension 1, dynamic programming does not work nearly as well in higher dimensions. One limitation on the efficiency of a dynamic programming algorithm is that one must, at some point, compute the value of each possible connected block. In dimensions 2 and higher, the worst-case complexity of dynamic programming will be exponential. In these dimensions, one can have a cell adjacent to each of the other $N-1$ cells and it will be contained in 2^{N-1} different connected blocks and any dynamic programming algorithm will have to compute the value of each of these blocks.

3. Algorithms for Partitioning Data in Higher Dimensions

In higher dimensions we also wanted to find an efficient algorithm for determining the optimal partition of a given set of data cells into blocks. In applications there are two related but distinct problems: Find the optimal partition of the data cells into

1. arbitrary blocks;

2. connected blocks.

In the former, the cells making up a block can lie anywhere in the data space, whereas in the latter, they must form a connected region. We say that a block B is connected, if and only if for any two cells c, d in B there is a sequence of cells $c = c_0, c_1, c_2, \ldots, c_m = d$ in B such that any two consecutive cells c_i, c_{i+1} are adjacent, $i = 0, 1, \ldots, m-1$. Contour maps provide an analogy. In the analog of (1) the levels may contain any number of contours that correspond to the same value. In the analog of (2), contour curves for the same level that do not intersect each other are considered distinct. In principle, the two problems can be quite different. In practice, the main difference is that in (1) regions of the data space widely separated from each other can combine their statistical weight to make structures that in (2) would have a smaller statistical significance, since the components of a disconnected block would be treated as separate smaller blocks and thus given less overall weight. For many applications it is appropriate to consider all possible partitions of the data cells into blocks (connected or not). In Figures 2 and 3 we see two different optimal partitions of the counties of California with respect to population density (persons per square mile). In Figure 2 we have the optimal partition into connected blocks (7 blocks) and in Figure 3 we have the optimal partition into arbitrary blocks (5 blocks). We will exhibit an $O(N^2)$ dynamic programming algorithm for finding the optimal partition into arbitrary blocks. This algorithm also extends to a branch and bound algorithm that can be used to find an optimal partition of the data into connected blocks. Because the worst-case complexity of the branch and bound algorithm is exponential, it is difficult to find the optimal partition into connected blocks for any large problem.

Let C be any set of N cells which cover the data space X. Let P be any partition of the data cells into blocks $B_1, B_2, \ldots, B_M, M \leq N$, that are connected unions of cells. Define P_{conn} to be the set of all such partitions of X. Similarly, we define P_{arb} to be the set of partitions of X into arbitrary blocks (not necessarily connected). Since we start with a finite number of cells then the number of partitions in P_{arb} (P_{conn}) is also finite. According to the intermediate density property (see below) to find the optimal partition of C into arbitrary blocks we only need to sort the cells by their densities. For an optimal partition, every block B is the union of consecutive cells from this sorted array. For every pair C_i and C_{i+1} in the sorted array we define C_i to be adjacent to C_{i+1}, for $i = 1, 2, \ldots, N-1$. Thus we apply the 1-d dynamic programming algorithm to the cells in the sorted array to find an optimal partition into arbitrary blocks in $O(N^2)$ time. This algorithm works for data in any dimension. See Figure 4 for the optimal partition of the data cells (Voronoi cells) of Figure 1 into arbitrary blocks and Figure 3 for the optimal partition of the California counties into

arbitrary blocks with respect to 1980 population density (persons per square mile).

In order to apply a branch and bound algorithm in finding an optimal partition of P_{conn} we need to be able to find ways of obtaining bounds on the value of a partition without actually computing it. We are searching for the optimal partition in P_{conn}, the set of partitions of the initial cells into connected blocks. To employ the branch and bound technique we expand our search to a larger class of problems. We will first search for the optimal partition P in P_{arb}, the set of partitions of the initial set C of N cells into arbitrary blocks.

Below we list the steps of our branch and bound algorithm for finding the optimal partition of C in P_{conn}. The set S is a set of open subproblems that starts with a single problem, that of finding the optimal partition of C in P_{arb}. Initially *bestvalue* has a value of negative infinity and as the algorithm progresses, *bestvalue* stores the largest value of a partition in P_{conn} that has been discovered so far.

1. For some problem T in S, we find the optimal partition P in P_{arb}.

2. If all the blocks of the optimal partition are connected, we say that P is a possible optimal solution (POS). Even if the optimal partition P has disconnected blocks then the value of P is an upper bound on the value of an optimal partition in P_{conn}, since P_{conn} is contained in P_{arb}. This is the "bounding" part of the branch and bound algorithm. If the value of P, $g(P)$, is less than or equal to *bestvalue* then T is removed from S since it cannot lead to a POS with a higher value. If $g(P)$ is greater than *bestvalue*, we define $bestvalue = g(P)$. Again T is removed from S and any other subproblem whose upper bound is less than or equal to $g(P)$ is also removed from S. If S is empty, then *bestvalue* is the optimal value of a partition in P_{conn} and the corresponding partition is an optimal partition, so we stop. If S is nonempty, we continue by returning to step 1 to look at another open problem in S.

3. If P has disconnected blocks we branch about a pair of adjacent cells i and j. Usually we let i be some cell in a disconnected block and let j be an adjacent cell outside of this block. We consider two subproblems, $T1$, where cells i and j are merged (to form a single cell), and $T2$, where cells i and j are separated (the adjacency between cells i and j is removed). Note that the optimal solution of $T1$ will be the optimal partition in P_{arb} with i and j in the same block. In the optimal solution of $T2$, cells i and j will not be merged directly. To avoid redundancy in the branch and bound algorithm one should not consider any future branches which involve merging a pair of cells that result in a cell that contains both i and j since this possibility has already been considered when i

and j were merged. We remove T from S and add the two new problems $T1$ and $T2$. We continue by returning to step 1 to look at another open problem in S. This is the "branching" part of the branch and bound algorithm.

Eventually every subproblem in S will end up with an associated optimal partition in P_{conn} since we can only branch on an adjacency between two cells once and after branching on every pair of adjacent cells we end up with a partition consisting of nonadjacent connected blocks. The corresponding optimal partition is this partition, which is in P_{conn}. Thus the branch and bound algorithm terminates when every subproblem is closed and the best POS discovered so far up to that point is now shown to be optimal. The worst-case complexity of this algorithm is at most $O(2^M)$, where M is the number of adjacencies between the original data cells. In fact, if we are careful to avoid redundancy as described in the third step above we see that this algorithm is $O(2^N)$, where N is the number of original number of data cells. If the branch and bound algorithm is implemented properly we expect that the average complexity is much better than this worst-case complexity. See Figure 5 for the optimal partition of the data cells (Voronoi cells) of Figure 1 into connected blocks and Figure 2 for the optimal partition of the California counties into connected blocks with respect to 1980 population density (persons per square mile).

4. Intermediate Density Property

To implement the branch and bound algorithm described above efficiently, we use something that we call the intermediate density property. The intermediate density property allows the one-dimensional dynamic programming algorithm to be used to find the optimal partition of the data into arbitrary blocks (not necessarily connected), even when the data comes from a higher dimension. This property says that if P_{max} is an optimal partition of a collection of cells into arbitrary blocks, with cells c and d in block B, and if e is a cell with density intermediate to the densities of cells c and d, then e must also be in block B. The proof of the intermediate density property uses the strict convexity of the function g that assigns a value (likelihood) to each of the blocks of a partition. If cell e is not in block B as described above, then the convexity allows us to find a better partition, contradicting the fact that P_{max} is optimal.

Definition: We say that a function $g(x, y)$ is strictly convex on a region X if and only if for any $0 < \lambda < 1$, and every pair of points (x_1, y_1), (x_2, y_2) in X,

$$\lambda g(x_1, y_1) + (1 - \lambda) g(x_2, y_2) \geq g(\lambda x_1 + (1 - \lambda) x_2, \lambda y_1 + (1 - \lambda) y_2) \qquad (5)$$

with strict inequality holding unless $x_1 = x_2$ and $y_1 = y_2$.

Let $C = C_1, C_2, \ldots, C_N$ be a set of cells partitioning the data space X, and let P represent a partition of the cells into M blocks, $B_1, B_2, \ldots, B_M$. We usually assume that each cell has 1 data point and thus the population of a block is equal to the number of cells that it contains. Suppose we want to find the optimal partition P_{max} in P_{arb} where blocks are allowed to be an arbitrary union of cells (not necessarily connected). We use the logarithmic form of the objective function

$$\begin{aligned} g(x,y) &= log[f(x,y)] && (6)\\ &= log[\beta(x-y+1, y+1)] && (7)\\ &= log[\int_0^1 p^{x-y}(1-p)^y dp] && (8) \end{aligned}$$

to compute the value of a partition P. Thus the value of P, $W(P)$ is $\sum g(a(B), n(B))$ where the sum is taken over all the blocks B in P. The density of block B is defined to be its population divided by its area, $d(B) = n(B)/a(B)$. The following result is what we call the intermediate density property.

The Intermediate Density Property: Let P_{max} be a partition in P_{arb} that maximizes W. Let B be any block in P_{max} and let C_1, C_2, C_3 be cells in C with C_1 and C_3 in B. If $d(C_1) < d(C_2) < d(C_3)$ then C_2 is also in B.

Let $C = C_1, C_2, \ldots, C_N$ be the data cells covering the data space X, sorted by their densities so that

$$d(C_1) \leq d(C_2) \leq \ldots \leq d(C_N). \quad (9)$$

The intermediate density property implies that for an optimal partition P_{max}, every block B in P_{max} is the union of consecutive cells from C. Thus to find an optimal partition in P_{arb} we only need sort the cells by their densities and then defining C_i to be adjacent to C_{i+1}, for $i = 1, 2, \ldots, N-1$, we apply the 1-d dynamic programming algorithm to these cells in order to find an optimal partition into arbitrary blocks. Since the same function g is used to assign values for a block no matter what dimension the data comes from, then this algorithm can be applied to find the optimal partition into arbitrary blocks regardless of the dimension of the data. If the blocks of a partition are required to be connected then the branch and bound algorithm will have to be used to find the optimal partition.

To prove the intermediate density property, we use several lemmas. First we prove (Lemma 1) that the function g which assigns a value to each of the blocks in a partition

is strictly convex, using Holder's inequality. Then we use several properties of a strictly convex function to complete the proof of the intermediate density property.

Lemma 1: The function $g(x,y) = log[f(x,y)] = log[\beta(x-y+1, y+1)] = log[\int_0^1 p^{x-y}(1-p)^y dp]$ is strictly convex on the region $X = \{(x,y)|x \geq y \geq 0\}$.

Proof of Lemma 1: To show that g is strictly convex we need to show that for any $0 < \lambda < 1$, and every pair of points (x_1, y_1), (x_2, y_2) in X, $\lambda g(x_1, y_1) + (1-\lambda) g(x_2, y_2) \geq g(\lambda x_1 + (1-\lambda)x_2, \lambda y_1 + (1-\lambda)y_2)$, with strict inequality holding unless $x_1 = x_2$ and $y_1 = y_2$. Note that

$$
\begin{aligned}
& g(\lambda x_1 + (1-\lambda)x_2, \lambda y_1 + (1-\lambda)y_2) && (10)\\
= \;& log(f(\lambda x_1 + (1-\lambda)x_2, \lambda y_1 + (1-\lambda)y_2)) && (11)\\
= \;& log(\int_0^1 p^{\lambda(x_1-y_1)+(1-\lambda)(x_2-y_2)}(1-p)^{\lambda y_1 + (1-\lambda)y_2} dp) && (12)\\
= \;& log(\int_0^1 [p^{\lambda(x_1-y_1)}(1-p)^{\lambda y_1}][p^{(1-\lambda)(x_2-y_2)}(1-p)^{(1-\lambda)y_2}] dp) && (13)\\
= \;& log(\int_0^1 [p^{(x_1-y_1)}(1-p)^{y_1}]^{\lambda}[p^{(x_2-y_2)}(1-p)^{y_2}]^{1-\lambda} dp) && (14)\\
\leq \;& log([\int_0^1 p^{x_1-y_1}(1-p)^{y_1} dp]^{\lambda}[\int_0^1 p^{x_2-y_2}(1-p)^{y_2} dp]^{1-\lambda}) && (15)\\
= \;& \lambda log(f(x_1,y_1)) + (1-\lambda) log(f(x_2,y_2)) && (16)\\
= \;& \lambda g(x_1,y_1) + (1-\lambda) g(x_2,y_2). && (17)
\end{aligned}
$$

The inequality in Lemma 1 follows from Holder's Inequality.

Holder's Inequality: For any nonnegative functions $A(x), B(x)$ and real numbers p, q such that for some $0 < \lambda < 1$, $p = 1/\lambda$ and $q = 1/(1-\lambda)$ (equivalently $1/p + 1/q = 1$), we have the following inequality:

$$\int_0^1 A(x)B(x)dx \leq [\int_0^1 A(x)^p dx]^{\lambda}[\int_0^1 B(x)^q dx]^{1-\lambda}, \qquad (18)$$

with equality holding if and only if $A(x)^p / B(x)^q$ is constant almost everywhere on $[0,1]$.

To prove the inequality in Lemma 1 note that if $A(x) = F(x)^{\lambda}$ and $B(x) = G(x)^{1-\lambda}$, then

$$\int_0^1 F(x)^\lambda G(x)^{1-\lambda} dx \tag{19}$$

$$\leq \; [\int_0^1 [F(x)^\lambda]^p dx]^\lambda \cdot [\int_0^1 [G(x)^{1-\lambda}]^q dx]^{1-\lambda} \tag{20}$$

$$= \; [\int_0^1 F(x) dx]^\lambda \cdot [\int_0^1 G(x) dx]^{1-\lambda}, \tag{21}$$

with equality holding if and only if $F(x)/G(x)$ is constant almost everywhere on $[0, 1]$.

Lemma 2: For any positive reals m, n_1, n_2 (we assume that $m - n_2 + 1 > n_1 - 1$, the function $h(x) = g(x, n_1) + g(m - x, n_2)$ is strictly convex on $I = (n_1 - 1, m - n_2 + 1)$.

Proof of Lemma 2: First we note that $g(x, n_1)$ and $g(m - x, n_2)$ are both strictly convex by Lemma 1. It is easy to show that the sum of two strictly convex functions is strictly convex.

Lemma 3: If $h(x)$ is a strictly convex function on $(a, b) \subseteq \Re$, and δ_1, δ_2 are positive real numbers such that $\{x - \delta_1, x + \delta_2\} \subseteq (a, b)$, then either $h(x - \delta_1) > h(x)$ or $h(x + \delta_2) > h(x)$.

Proof of Lemma 3: Assume $h(x - \delta_1) \leq h(x)$. Since h is strictly convex,

$$h(x) < [\delta_2/(\delta_1 + \delta_2)]h(x - \delta_1) + [(1 - (\delta_2/(\delta_1 + \delta_2))]h(x + \delta_2). \tag{22}$$

Multiplying both sides of this inequality by $\delta_1 + \delta_2$ we get

$$\delta_1 h(x) + \delta_2 h(x) < \delta_2 h(x - \delta_1) + \delta_1 h(x + \delta_2). \tag{23}$$

Then since $h(x - \delta_1) \leq h(x)$, it must be that $h(x + \delta_2) > h(x)$. By similar reasoning, if $h(x + \delta_2) \leq h(x)$, then $h(x - \delta_1) > h(x)$.

Proof of the Intermediate Density Property: Let P_{max} be a partition of C that maximizes W. Let blocks B_1 and B_2 be any pair of different blocks in P, and let C_1, C_2, C_3 be cells in C, with $\{C_1, C_3\} \subseteq B_1$ and $d(C_1) < d(C_2) < d(C_3)$. Assume for contradiction that C_2 is in B_2. If each cell contains a single data point then $a(C_3) < a(C_2) < a(C_1)$. Thus $\delta_1 = a(C_2) - a(C_3) > 0$ and $\delta_2 = a(C_1) - a(C_2) > 0$. We now consider two new partitions P_1 and P_2 created by swapping cell C_2 for each of C_1, C_3 in B_1. Let

$$P_1 = (P - \{B_1, B_2\}) \cup \{B_1', B_2'\} \tag{24}$$

and

$$P_2 = (P - \{B_1, B_2\}) \cup \{B_1'', B_2''\} \tag{25}$$

where

$$B_1' = (B_1 - \{C_3\}) \cup \{C_2\}, \tag{26}$$

$$B_2' = (B_2 - \{C_2\}) \cup \{C_3\}, \tag{27}$$

$$B_1'' = (B1 - \{C_1\}) \cup \{C_2\}, \tag{28}$$

$$B_2'' = (B_2 - \{C_2\}) \cup \{C_1\}. \tag{29}$$

Let P' be the partition $P_{max} - \{B_1, B_2\}$. The value of partition P_{max} in terms of $h(x) = g(x, n(B_1)) + g(a(B_1) + a(B_2) - x, n(B_2))$ is

$$\begin{aligned} W(P_{max}) &= \sum_{B \in P_{max}} g(a(B), n(B)) && (30)\\ &= g(a(B_1), n(B_1)) + g(a(B_2), n(B_2)) + \sum_{B \in P'} g(a(B), n(B)) && (31)\\ &= h(a(B_1)) + W(P'). && (32) \end{aligned}$$

Similarly $W(P_1) = h(a(B_1) - \delta_1) + W(P')$ and $W(P_2) = h(a(B_1) + \delta_2) + W(P')$. By Lemma 2, $h(x)$ is convex and by Lemma 3, either $h(a(B_1) - \delta_1) > h(a(B_1))$ or $h(a(B_1) + \delta_2) > h(a(B_1))$. Thus either $W(P_2) > W(P_{max})$ or $W(P_1) > W(P_{max})$ contradicting the fact that P_{max} maximizes W. Therefore C_2 is not in B_2 and since B_2 is an arbitrary block different from B_1, it must be that $C_2 \in B_1$.

In (Scargle 1998) we also have the following global likelihood for data that is prebinned into evenly spaced intervals (where Λ represents the constant rate per bin),

$$\int_0^\infty \Lambda^N e^{(-M+1)\Lambda} d\Lambda = \Gamma(N+1)/(M+1)^{N+1} \tag{33}$$

for a block of N data points in M bins. For prebinned data, the data cells in the starting partition are taken to be the bins which can contain any number of data points. As before the likelihood of a partition is assumed to be the product of the likelihoods of its blocks and taking the logarithm we get an objective function that satisfies the additive property.

Also in (Scargle 1998) we have a similar likelihood function for time to spill (TTS) data on an interval. Assuming only every Sth photon is recorded and that $\tau_1, \tau_2, \ldots, \tau_{n-1}$ are the lengths of the data cells (intervals between spill events) then the likelihood that the intensity is constant over a block is

$$[(\prod_{n=1}^{N-1} \tau_n)^{S-1}/\Gamma(S)^{N-1}] \cdot [\Gamma(S(N-1)+1)/(M+1)^{S(N-1)+1}] \tag{34}$$

where $M = \sum_{n=1}^{N-1} \tau_n$ is the length of this block and $S(N-1)$ is equal to the number of data points in this block. The likelihood for a partition of data cells into blocks is thus a constant (depends only on S and N and not on the partition), multiplied by a function that is equal to the likelihood function for the binned data.

Note that the proof of the intermediate density property given here requires that the number of data points in each cell is 1. It seems that a similar proof (though slightly more complicated) shows that the intermediate density property is still true for an arbitrary set of data cells (cells can contain any number of data points). A proof quite similar to that in Lemma 1 shows that the likelihood function for binned data is strictly convex as well and since the likelihood function for binned data is strictly convex we see that the likelihood function for TTS data is also strictly convex. We deduce that the intermediate density property holds for both binned data and TTS data. Thus the algorithms described in this paper can be used to find the optimal partitions for data in equal-spaced bins and for TTS data as well.

5. Dynamic Programming on Sorted Data

Another extension of the intermediate density property shows that if two cells of the starting partition are equal in density, then they are in the same block of the optimal partition. This fact can be used to reduce the complexity of the dynamic programming to $O(D^2)$, where D is the number of different densities of the data cells. For example, in a grayscale image, each pixel initially receives a shade of gray from 1 (white) to 256 (black). We have used our algorithm to compress the image of a grayscale image by reducing the number of colors required. Each pixel is assigned to a block and each block receives a color (usually the number of blocks is far fewer than 256). This optimal compression can be found in $O(256^2)$ time as opposed to $O(P^2)$ time, where P is the number of pixels (P is usually much larger than 256).

In comparing our techniques for partitioning data with some of the standard data clustering techniques for higher dimensional data, we note that our method again compares all partitions of the data, regardless of the number of blocks. The standard techniques for clustering a set of data points (Alpert,Kahng 1997) into k clusters, so that the maximum cluster diameter (or the sum of the cluster diameters) is minimized, require the number of clusters to be fixed ahead of time. For dimension 2 and higher it is known that these standard problems are NP-complete (Garey,Johnson

1979). We have presented an $O(N^2)$ algorithm for finding the optimal partition of N data points into arbitrary blocks, that works for data in any dimension. However, we don't yet know if there is an efficient algorithm for finding an optimal partition of the data into connected blocks. The complexity of the branch and bound algorithm we described earlier, for finding the optimal partition of a set of data points into connected blocks, is exponential. We suspect that this problem is NP-complete in dimension 2 and higher, but we have not yet been able to prove it.

REFERENCES

Alpert, C. J. and Kahng, A. B., Splitting Orderings into Multi-way Partitionings to Minimize the Maximum Diameter, Journal of Classification, (14), 1997, pp. 51-74.

Barry, D. and Hartigan, J.A., Product partition models for change point problems, J. Amer. Statist. Assoc., 20, 1992, 260-279.

Bellman, R., Dynamic Programming, Princeton University Press, Princeton, 1957.

Garey, M. and Johnson, D., Computers and Intractability W.H. Freeman and Company, New York, 1979.

Hubert, P., Change points in meteorological time series, Applications of Time Series Analysis in Astronomy and Meteorology, Rao, T., Priestly, M., and Lessi, O., eds., 1997, Chapman and Hall.

Jackson, B., Scargle J., et. al., Optimal Partitions of Data on an Interval, accepted for publication in IEEE Signal Processing Letters.

Kay, S. M., Fundamentals of Statistical Signal Processing: Detection Theory, Englewood Cliffs. NJ: Prentice-Hall, 1998.

Kehagias, A., Nicolau, A., Fragkou, P., Petridis, V., Text Segmentation by Product Partition models and Dynamic programming, Mathematical and Computer modeling, 39, 2004, 209-217.

Lee, Peter M., Bayesian Statistics: An Introduction, 2nd edition, Arnold, London, 1997.

Quintana, F., and Iglesias, P., Bayesian Clustering and Product Partition Models, Journal of the Royal Statistical Society, Series B, 65, 557-574, 2003.

Scargle, J., Studies in Astronomical Time Series Analysis. V. Bayesian Blocks, A New Method to Analyze Structure in Photon Counting Data, The Astrophysical Journal, (504), 1998, pp. 405-418.

Vidal, R., Optimal Partition of an Interval, Applied Simulated Annealing, Springer-Verlag, New York, 1993, 277-291.